CODE CLIMATE

Sample Customer Insights Report

About the customer: A leading provider in payroll services, HR consulting, and insurance. Please note that all sensitive customer information has been anonymized.

Executive Summary

The customer desires more visibility into the challenges and opportunities limiting productivity within a subsidiary. The Director of Product Development, on behalf of their organization, engaged Code Climate to analyze quantitative data available through BitBucket & Jira, and share insights into bottlenecks and opportunities.

Code Climate found several insights, with a few worth highlighting:

- **Through the lens of Pull Request (PR) Cycle Time, productivity looks low– largely due to long Time to Open cycles**. Our research shows that PRs opened early remain open for less time. Related to this, we found a high volume of large PRs (over 400+ lines), which tend to take longer to review.
    - **Recommendation:** The cause of these large PRs and long time to open is worth additional inspection, but effective strategies to reduce both include requiring that developers reduce the size of their PRs, and open them soon after their first commit. Additionally, ensuring developers are writing clear descriptions of the scope of their PRs at the outset may safeguard against scope creep.
- **Average Weekly Coding Days is extremely low at 1.9**. Several teams are close to 2 days/week, but Guardians and Power Squad are only averaging ~ 1 day/week. With industry average being closer to 4 days/week, this is cause for further inspection.
    - **Recommendation:** Are developers struggling with complexity or work? Are they spending a lot of time in meetings, vs developing? Are they burned out? The goal here is to improve weekly coding days, but there are many potential underlying issues so it's a matter of pinpointing the one most impacting this metric and testing a solution or two. Our Developer Experience survey is a means of illuminating what might be happening to contribute to this low weekly coding day value.
- **A few developers are pulling most of the weight:** Related to the above, most commits are coming from 3 developers and a few close seconds, but the majority of developers have extremely low commits per week. If these developers are not already burned out, they are likely at risk, and if they leave, productivity will drop further.
    - **Recommendation:** Are the highest contributing developers serving as a "crutch" for other developers? Consider whether these developers could play a more active role in supporting higher productivity of others, be it by enforcing small PRs and/or playing reviewers roles. Also consider enrolling additional support from people managers to coach lower contributors to improve their commits and overall throughput.

About this Report

This report provides a quantitative overview of the customer's engineering organization based on Code Climate's industry-leading software engineering intelligence expertise and benchmark data. Code Climate provides commentary on the data that's informed by looking at related metrics, historical trends, and the operating context provided by the customer during the qualitative interviews conducted by the Code Climate team. Engineering leaders can use the tailored insights provided in this report to select focus areas for their teams and design and operate experiments for improvement.

Audience

We prepared this report for the Director of Product Development, the Vice President, Platform and Technology Services, Sr. Director Software Engineering, and 3 Senior Engineering Managers.

Scope

We analyzed Bitbucket & Jira data from October 1st, 2024 through February 1, 2025 for 84 engineers in the subsidiary.

Next Steps

After digesting this report, you'll select 1-3 opportunities you'd like to tackle in partnership with us. Together, we'll devise & implement strategies to improve in these areas.

Overall Organization: Insights & Recommendations

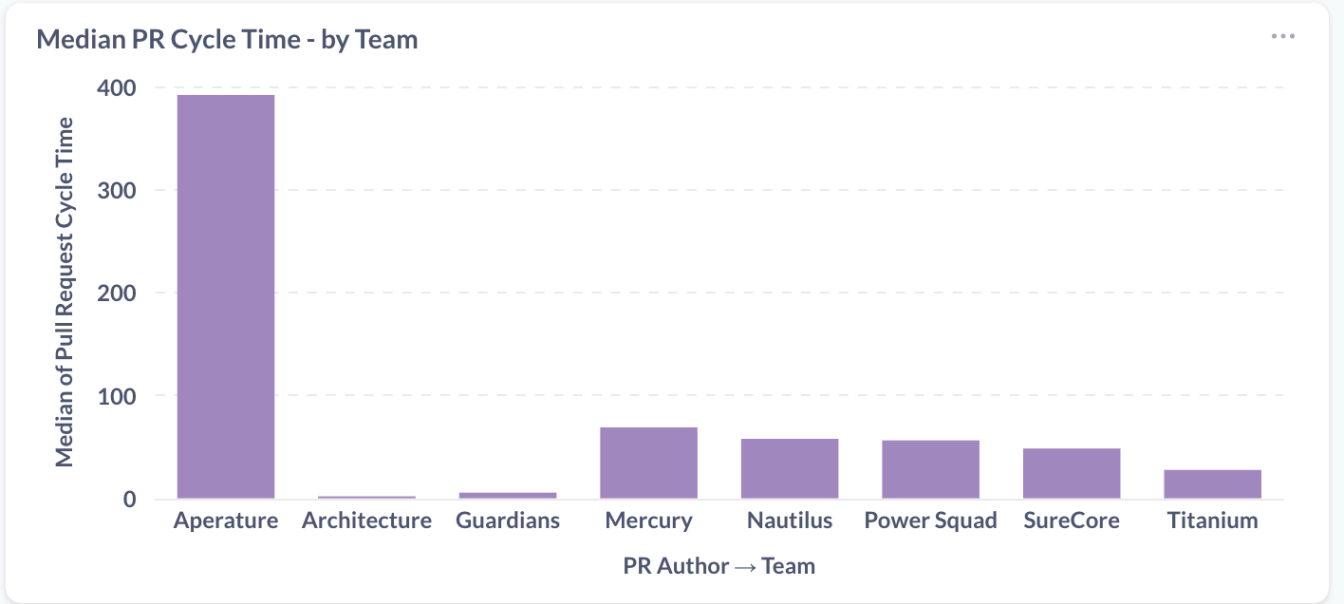
We benchmarked the subsidiary against the top 25% of the industry in three areas: **Productivity**, **Coding Process**, and **Review Process**. Benchmarks are derived from data from approximately 400 engineering organizations across multiple industries. Values are based on the most recent 12-month period, and only consider organizations with a significant amount of activity across that time. We also analyze the code contributions to exclude bot and open-source contributors, as well as use statistical methods to avoid atypical weeks and data points from skewing the benchmark values.

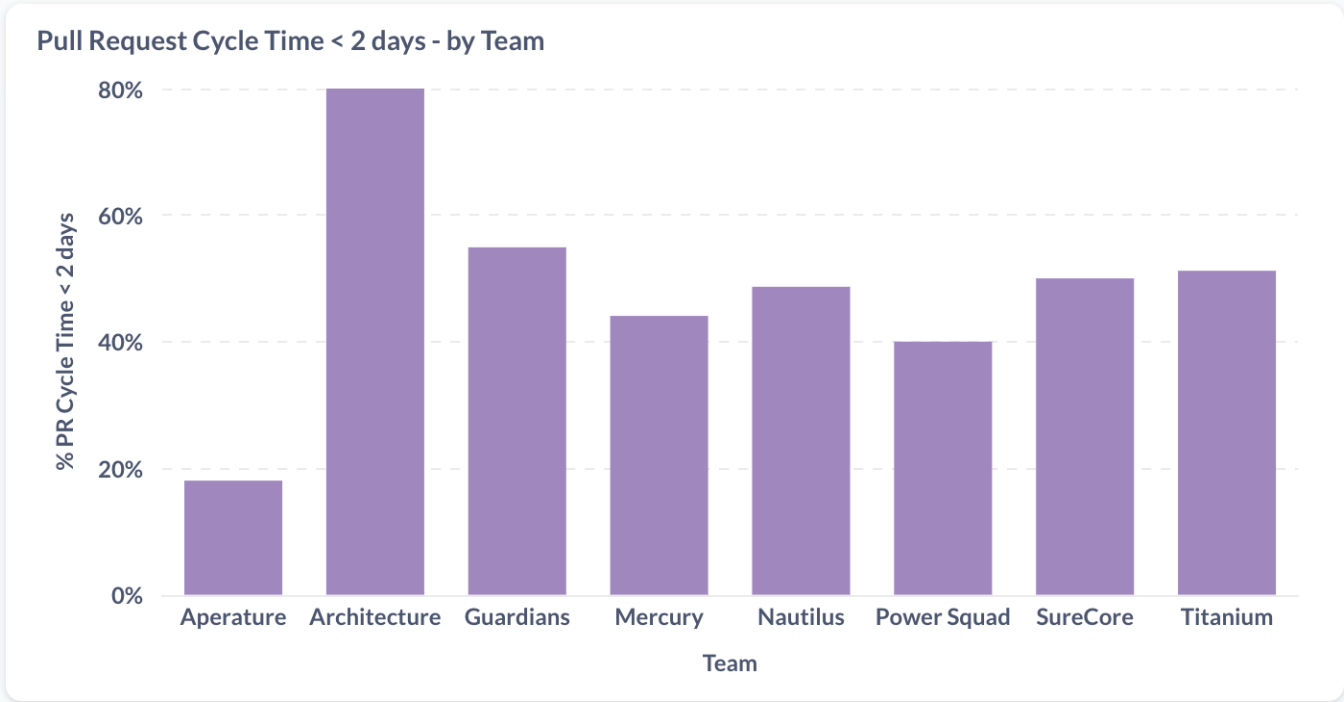
Productivity

Metric	Definition	Top 25% Benchmark	Subsidiary
Median PR Cycle Time (hours)	The median Pull Request Cycle Time, which is the time between when the first commit is authored to when a pull request is merged. Cycle Time starts at the first commit on the branch, vs. when the branch is created.	4 and lower	68.74 🔴
Pull Request Cycle Time < 2 Days	The percentage of pull request cycles that are shorter than 2 days	84% and higher	47% 🔴
Average Weekly PR Throughput per Engineer	The average weekly count of merged pull requests.	5.4 PRs and higher	2.89 🔴
Average Weekly Coding Days per Engineer	The average number of days per week that a developer on the team authors at least one commit.	3.5 days and higher	1.88 🔴

Analysis

All 4 metrics are well below the industry benchmark, and through further analysis we found a notable exception being the Architecture team which surpasses others in key measures:





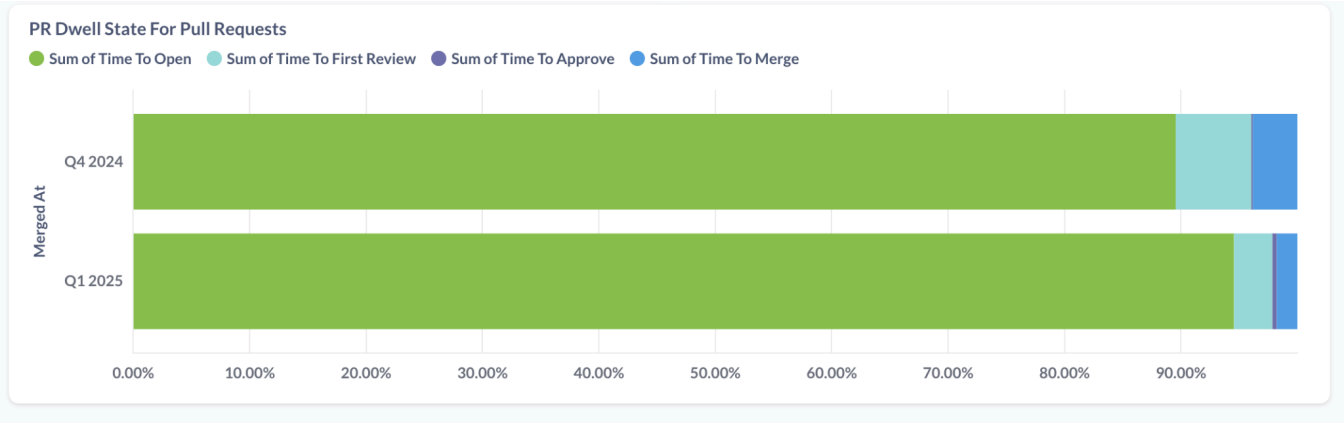
This confirms leadership's observations that productivity is lower than it could be within the subsidiary, and some potential clues emerge when exploring Coding & Review Process metrics: one key issue may be size and complexity of batches of work.

Coding Process

Metric	Definition	Top Quartile Benchmark	Subsidiary
PRs with 400+ Lines	Large pull requests.	10% and less	15% 🔴
PR Success Rate	The percentage of pull requests opened that are eventually merged.	92% and higher	95% 🟢
Time to Open < 0.5 Days	The time between the earliest commit in a pull request and when the pull request is created.	95% and higher	47% 🔴

Analysis

The vast majority of PRs complete their cycle– but as we saw in the previous section, PR cycles are long, with many of them lingering in Open state. This chart illustrates how much time PRs are “dwelling” in Open state, and then awaiting review:



Our theory is that some developers are adding too many commits to their PRs, which causes them to be unwieldy in scope, and prone to issues (which makes quick review difficult if not impossible). Governance to enforce short Time to Open cycles as well as keeping PRs small in scope could have a dramatic impact on throughput.

Review Process

Metric	Definition	Top Quartile Benchmark	Subsidiary
Defect Rate	The percentage of merged pull requests that are addressing defects.	14% and lower	7% 🟢
Time to First Review < 1 Day	The time between when a pull request is ready for review to when the first review is submitted.	93% and higher	84% 🔴
PRs with 3+ Review Cycles	% of times a pull request goes back and forth between the author and a reviewer more than 3 times.	2.5% and lower	2% 🟢

Analysis

Time to first review is relatively high; industry standard is to have most reviews occur within a day, and on average at the subsidiary it takes longer than that. Exploring further, we found that this is not isolated to a small handful of outliers. It could be that early reviews are not a typical part of the process for some developers,or this could be directly related to the large PR size issue we noted above. In any case, it would be useful for the subsidiary to promote short Time to First Review cycles to remediate any issues in code earlier in the process and improve overall throughput. Should the customer decide to continue the journey with Code Climate, we will partner to explore this and/or other top opportunities aligned with business goals further, and design + implement experiments for improvement.

Insights by Manager, Agile Team, Contractor vs. FTE, and Offshore vs. Onshore

After understanding overall organization performance vs. the above metrics, we explored the differences between HR reporting lines, agile teams, by employee status (contractor vs. FTE), and by location (offshore vs onshore). We found:

- Manager by Manager, we see some interesting differences in all metrics. Both have large teams with large swaths of FTE who are onshore, but the metrics are quite different. Is the challenge in the nature of the work, is it further upstream, and/or lack of time for more intentional coaching of staff?
- Team by team, we noticed **Architecture is outperforming other teams** in Productivity metrics (e.g. PR Cycle Time). We heard they are often jumping in and fixing things. Is it possible they are serving as a crutch for other teams who are less productive, such as Power Squad (lowest metrics in several areas)?
- **Offshore folks, on average, code 9% less per week than Onshore**, and have larger PRs. This caused us to wonder about communication patterns between Offshore and Onshore,

and once again think about the clarity of description per PR as well as the value of small PRs.

- **Contractors, on average, code 14% more per week than FTE**. Their cycle times are longer as well. Deeper analysis suggests this may correlate with them also having more PRs with 3+ reviews than FTEs; possible causes explored below.

Manager by Manager

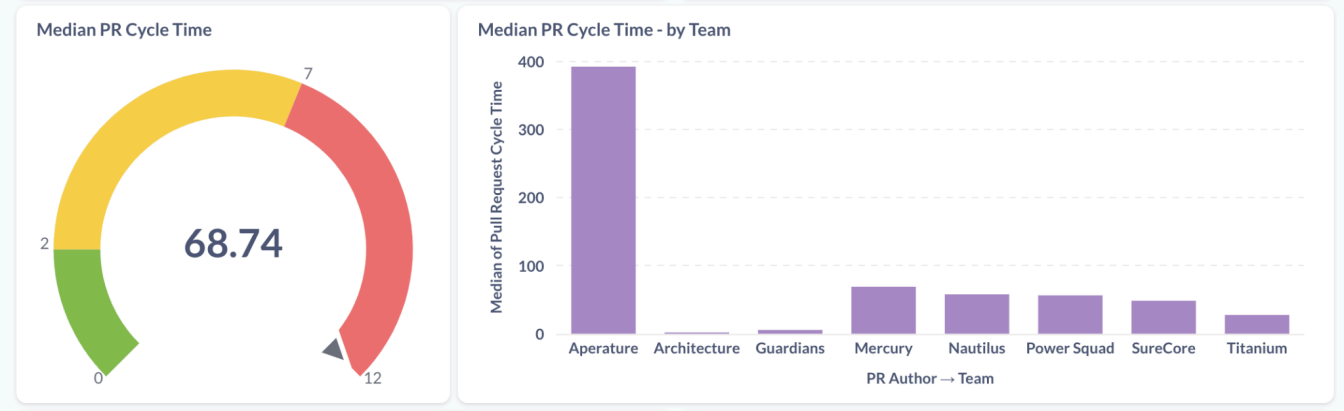
Name	Contributors	Median Time to Open	Median PR Cycle Time	Defect Rate	Avg. Weekly Coding Days per Dev
Manager 1	7	25.29 hrs	56.2 hrs	13.3 %	1.2
Manager 2	10	1.70 hrs	57.7 hrs	9.3 %	1.9
Manager 3	10	17.17 hrs	27.5 hrs	7.2 %	1.9
Manager 4	13	**0.25 hrs**	**5.5 hrs**	**0.0 %**	1.0
Manager 5	2	0.07 hrs	0.4 hrs	**40.0 %**	-
Manager 6	16	23.41 hrs	48.4 hrs	6.5 %	2.0
Manager 7	2	-	-	-	-
Manager 8	13	**215.43 hrs**	**391.7 hrs**	9.8 %	2.2
Total (across managers)		20.55 Hours	68.7 Hours	7.4 %	1.9
Organization (across subsidiary)		21.98 Hours	68.7 Hours	7.1 %	1.9
Industry Benchmark		0.05 Hours	4.0 Hours	15.0 %	3

Insights & Actions

- **The high defect rate for Manager 5's team may be worth inspecting further**. Defect rate equates to PRs resolving defects, and is an indicator of a high quantity of defects. Manager 5 runs the Architecture team, and given context we took in previously about how this team is often pulled in to troubleshoot and fix, this makes sense– but we'll reference our earlier consideration about whether this is causing other developers to be less rigorous and/or transparent with respect to their code, earlier in the process.
- **Highs and lows, worth more context/exploration**: Manager 4's team metrics are stronger than most in many areas except for Coding Days. Trevor West's team metrics are concerning, but perhaps challenges exist further upstream? This is worth closer inspection.

By Agile Team

Our key finding here is that Architecture is performing much better than other teams in several metrics, particularly those that indicate Productivity. Another thing that stood out to us is how the Aperature team's longer PR cycles is impacting the overall Median, as illustrated here (the unit is hours):

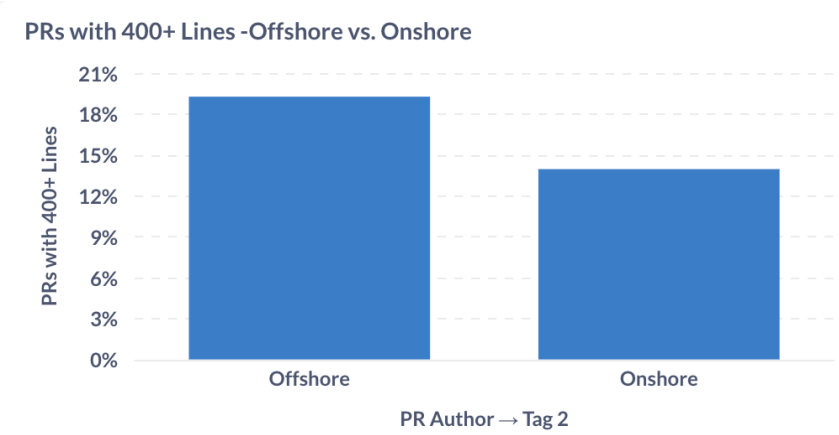


Insights & Actions

- "Architecture tends to come in and save the day" is what we heard in discussions prior to the diagnostic. We'd recommend considering whether teams are ***relying*** on Architecture to correct issues and, perhaps as a result, are less stringent about things that could speed up the delivery of solutions, like opening PRs early with clear descriptions and small scope. What might happen if Architecture was not available for 2 months, and there was governance and coaching to ensure PRs were small and well defined?

Offshore vs. Onshore

Offshore contributors are slightly less productive than onshore contributors, with a larger percentage of XL PRs:



From discussions with the Director of Product Development, we understand that the majority of offshore people are working on automated test suites and they tend to copy, paste, and tweak existing suites to create new ones. This could be a large contributing factor to their large PRs. There may be other things at play, too– e.g. missing context.

Insights & Actions

To improve productivity of offshore staff, we'd recommend probing further into the potential reasons why this might be. If it is something as simple as requiring smaller PRs that are tighter in scope, this could be a relatively simple way to improve productivity.

Contractor vs. FTE

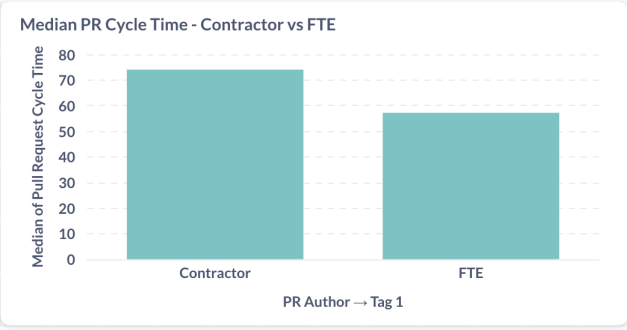
Contractors, we found, code more on a weekly basis than FTE employees– which, in the absence of additional context, could look like a good thing– except we also saw their PRs are longer, so value is delayed.

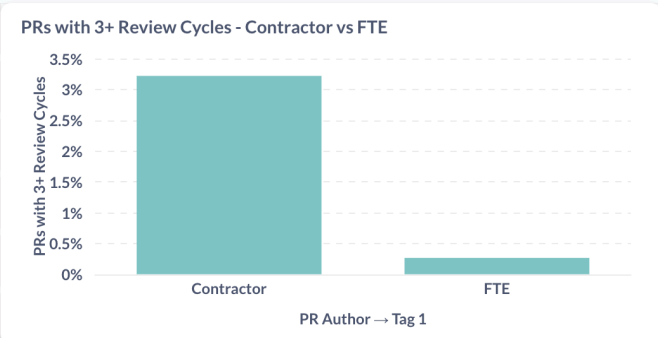
Insights & Actions

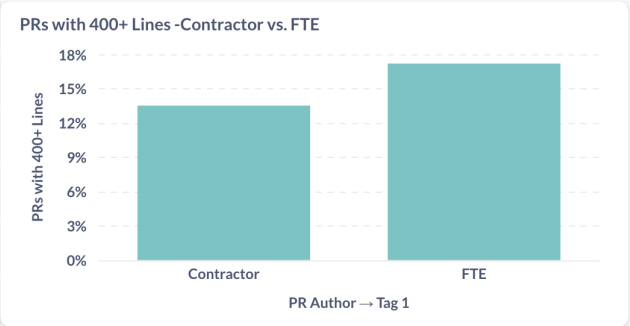
Digging deeper into this, we found that Contractors are more likely to have PRs with 3+ reviews than FTEs, which tends to mean PRs are being corrected and then re-reviewed. Their PRs tend to be smaller than FTEs, too, so it is less likely that the reason for so many reviews is that they are updating AND adding to PRs. Note that the percentage of PRs needing 3+ reviews is quite low, so not cause for huge alarm and this wouldn't be the opportunity we'd suggest choosing to tackle first– but interesting nonetheless.

Worth exploration: What is the reason for most frequent reviews? Reviews are a good practice; is this too much of a good thing? Or are there other reasons that may have more to do with skill/lack of context that

in turn show up in code that has issues that need to be resolved prior to merge?







Conclusion

Improving Time to Open should be “low hanging fruit” for the subsidiary to improve productivity, as is requiring smaller PR sizes. Another opportunity to make a big impact is in supporting a more even distribution of work, given the relatively low volumes of commits for several developers. As we performed the above analysis, and given the patterns we saw, we also wondered about the complexity of the work being performed, and whether some of the challenges may be further upstream. By improving in these metrics, the subsidiary team will be more productive, with higher quality code – but the end goal is to deliver more **business value**, faster, and make a tangible impact on business goals. Are PRs building in size because of changing or unclear requirements, and/or “definition of done?” This might also explain long Time to Open cycles. Questions like these, and others offered in this report, are worth additional exploration– and we’d be delighted to continue the journey with you, moving quickly from insight, to diagnosis, to action. We hope that you find this analysis useful and actionable, and look forward to discussing this further with you.

Appendix

About Code Climate

Code Climate was founded in 2011 by engineering leaders who were frustrated by the lack of visibility into data that would enable them to make informed improvements. Today, engineering executives from Fortune 100 companies in industries such as Hospitality, Healthcare, Retail, and Technology partner with Code Climate to tackle critical business challenges. Through our personalized approach, expert guidance, and a custom-designed Software Engineering Intelligence (SEI) platform, Code Climate delivers tailored, actionable insights that empower engineering executives to make data-driven decisions.

Continuing the Journey

With this report, you’ve gained context to identify and tackle significant opportunities for improvement in your organization. Now, it’s imperative to take action.

We believe value only comes when engineering leaders act on important insights. We also believe value sooner is better– and quick, light experiments are the best means to effect measured improvement week over week, month over month. Given our background in Lean Startup, we're experienced at helping organizations become more productive, higher performing, and thriving.

We look forward to discussing the findings of this analysis in more detail with you, and understanding what stands out to you as the most valuable and important opportunities to tackle first. We'll share with you a scorecard we've developed for you to monitor key metrics, and identify opportunities to improve by drilling deeper into the data. From there, we'll begin forming hypotheses: What problem do we observe? What do we believe is the root cause? What do we believe could solve for this, quickly– and what metric will we see improve to indicate we're right? We run experiments to deliver measured improvement, and to learn how to frame the next experiment to deliver even better results.